

Extending Partial Representations of Subclasses of Chordal Graphs

Pavel Klavík*, Jan Kratochvíl*, Yota Otachi**, and Toshiki Saitoh***

Abstract. Chordal graphs are intersection graphs of subtrees in a tree. We investigate complexity of the partial representation extension problem for chordal graphs. A partial representation specifies a tree T' and some pre-drawn subtrees. It asks whether it is possible to construct a representation inside a modified tree T which extends the partial representation (keeps the pre-drawn subtrees unchanged).

We consider four modifications of T' and get vastly different problems. In some cases, the problem is interesting even if just T' is given and no subtree is pre-drawn. Also, we consider three well-known subclasses of chordal graphs: Proper interval graphs, interval graphs and path graphs. We give an almost complete complexity characterization.

Also, we study parametrized complexity by the number of pre-drawn subtrees, the number of components and the size of the tree T' . We describe an interesting relation with integer partition problems. The problem 3-PARTITION is used in the NP-completeness reductions. The BINPACKING problem is closely related to the extension of interval graphs when space in T' is limited, and we obtain “equivalency” with BINPACKING.

1 Introduction

Geometric representations of graphs and graph drawing are well-studied topics in graph theory. We study *intersection representations* of graphs where the goal is to assign geometrical objects to the vertices of the graph and encode edges by intersections of the objects. An intersection-defined class restricts the geometrical objects and contains all graphs representable by these restricted objects; for example, interval graphs are intersection graphs of closed intervals of the real line. Intersection-defined classes have many intersecting properties and appear naturally in numerous applications; for details see for example [9].

For a fixed class, its recognition problem asks whether an input graph belongs to this class; in other words, whether it has an intersection representation of this

* Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Malostranské náměstí 25, 118 00 Prague, Czech Republic. E-mails: {klavik,honza}@kam.mff.cuni.cz. Supported by ESF Eurogiga project GraDR as GAČR GIG/11/E023.

** School of Information Science, Japan Advanced Institute of Science and Technology. Asahidai 1-1, Nomi, Ishikawa 923-1292, Japan. Email: otachi@jaist.ac.jp

*** Graduate School of Engineering, Kobe University, Rokkodai 1-1, Nada, Kobe, 657-8501, Japan. E-mail: saito@eedept.kobe-u.ac.jp

class. The complexity of recognition is for many classes well-understood; for example interval graphs can be recognized in linear-time [2, 4].

A recent paper [14] introduced the following new problem called *partial representation extension*. Given a graph and a partial representation (a representation of an induced subgraph), it asks whether it is possible to extend this representation to the entire graph. This problem falls into the paradigm of extending partial solutions, an approach that has been studied frequently in other circumstances. Often it proves to be much harder than building a solution from scratch, for example for graph coloring. Surprisingly, a very natural problem of partial representation extension was only considered recently.

The paper [14] gives an $\mathcal{O}(n^2)$ -algorithm for interval graphs and an $\mathcal{O}(nm)$ -algorithm for proper interval graphs. Also, several other papers consider this problem. Interval graphs can be extended in time $\mathcal{O}(n + m)$ [1]; proper interval graphs in time $\mathcal{O}(n + m)$ and unit interval graphs in time $\mathcal{O}(n^2)$ [13]; function and permutation graphs in polynomial time [12].

In this paper, we follow this recent trend and investigate complexity of partial representation extension for chordal graphs. Our mostly negative results are interesting since chordal graphs are the first class for which the partial representation problem becomes harder than the original recognition problem. Also, we investigate three well-known subclasses proper interval graphs, interval graphs and path graphs, for which the complexity results are more rich. We believe that better understanding of these simpler cases will give tools to attack chordal graphs and beyond (for example, from the point of parametrized complexity).

1.1 Chordal Graphs and Their Subclasses

A graph is *chordal* if it does not contain an induced cycle of the length four or more, i.e., each “long” cycle is triangulated. The class of chordal graphs, denoted by CHOR, is well-studied and has many wonderful properties. Chordal graphs are closed under induced subgraphs and contain so called *perfect elimination schemes*, close related to optimal ways for Gaussian elimination for sparse matrices. Chordal graphs are perfect and many hard combinatorial problems are easy to solve on chordal graphs: maximum clique, maximum independent set, k -coloring, etc. Chordal graphs can be recognized in time $\mathcal{O}(n + m)$ [16].

Chordal graphs have the following intersection representations. For every chordal graph G there exists a tree T and a collection $\{R_v : v \in V(G)\}$ of subtrees of T such that $R_u \cap R_v \neq \emptyset$ if and only if $uv \in E(G)$. For an example of a chordal graph and one of its intersection representations, see Figure 1.

When chordal graphs are viewed as *subtrees-in-tree* graphs, it is natural to consider two other possibilities: *subpaths-in-path* which gives *interval graphs* (INT), and *subpaths-in-tree* which gives *path graphs* (PATH). For example the graph in Figure 1 is a path graph but not an interval graph. This subpaths-in-path representations of interval graphs can be viewed as a discretizations of the real line representations. Interval graphs can be recognized in $\mathcal{O}(n + m)$ [2, 4] and path graphs in time $\mathcal{O}(nm)$ [8, 17].

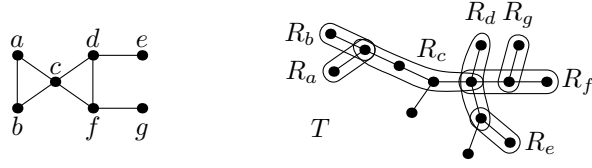


Fig. 1. An example of a chordal graph with one of its representations.

In addition, we consider proper interval graphs (PINT). An interval graphs is a proper interval graph if it has a representation \mathcal{R} for which $R_u \subseteq R_v$ implies $R_u = R_v$; so no interval is a proper subset of another. From point of our results, PINT behaves very similar to INT, but there are subtle differences which we consider interesting. Also, partial representation extension of PINT is surprisingly very closely related to partial representation extension of unit interval graphs considered in [13]; in details discussed below. Proper interval graphs can be recognized in time $\mathcal{O}(n + m)$ [15, 3].

1.2 Partial Representation Extension

For a class \mathcal{C} , we denote the recognition problem by $\text{RECOG}(\mathcal{C})$. For an input graph G , it asks whether it belongs to \mathcal{C} , and moreover we may certify it by a representation. The partial representation extension problem denoted by $\text{REPEXT}(\mathcal{C})$ asks whether a part of the representation given by the input can be extended to a representation of the whole graph.

A partial representation \mathcal{R}' of G is a representation of an induced subgraph G' . The vertices of G' are called *pre-drawn*. A representation \mathcal{R} extends \mathcal{R}' if $R_v = R'_v$ for every $v \in V(G')$. The meta-problem we deal with is the following.

Problem: $\text{REPEXT}(\mathcal{C})$ (Partial Representation Extension of \mathcal{C})

Input: A graph G with a partial representation \mathcal{R}' .

Output: Does G have a representation \mathcal{R} that extends \mathcal{R}' ?

In this paper, we study complexity of the partial representation extension problems for CHOR, PATH and INT in the setting of subtree-representations. Here the partial representation \mathcal{R}' fixes subtrees belonging to G' and also specifies some tree T' in which these subtrees are placed. The representation \mathcal{R} is placed in a tree T which is created by some modification of T' . We consider four possible modifications and get different extension problems:

- FIXED – the tree can not be modified at all, $T = T'$.
- SUB – the tree can only be subdivided; T is a subdivision of T' .¹

¹ Let an edge $xy \in E(T')$ be subdivided (with a vertex z added in the middle). Then also pre-drawn subtrees containing both x and y are modified and contain z as well. So technically in the case of subdivision, it is not true that $R'_u = R_u$ for every pre-drawn interval but from the topological point of view the partial representation is extended.

- ADD – we can add branches to the tree; T' is a subgraph of T .
- BOTH – we can both add branches and subdivide; a subgraph of T is a subdivision of T' , or in other words T' is a topological minor of T .

We denote the problems by $\text{REPEXT}(\mathcal{C}, \text{type})$. Constructing a representation in a specified tree T' is interesting even if no subtree is pre-drawn, i.e., G' is empty; this problem is denoted by $\text{RECOG}^*(\mathcal{C}, \text{type})$. Clearly, hardness of the RECOG^* problem implies the hardness of the corresponding REPEXT problem.

Concerning chordal graphs, the types ADD and BOTH allow to construct an arbitrary tree T , so the RECOG^* problem is equivalent to the standard RECOG problem. For interval graphs, the types ADD and SUB behave differently. The type ADD allows to extend the ends of the paths. The type SUB allows to expand the middle of the path but if the endpoint of the path is contained in some pre-drawn subpath, it remains there even after subdivision. The type BOTH is equivalent to the RECOG and REPEXT problems for the real line.

1.3 Our results

We consider complexity of the RECOG^* and REPEXT problems for all four classes and all four types. Our results are displayed in the table in Figure 2.

- All NP-complete results are reduced from the 3-PARTITION problem. The reductions are very similar and the basic case is $\text{REPEXT}(\text{PINT}, \text{FIXED})$.
- Polynomial cases for INT and PINT are based on the known algorithm for recognition and extension. But since the space in T is limited, we adapt the algorithm for the specific problems.

Also, we study parametrized complexity of these problems with respect to three parameters: The number of pre-drawn subtrees k , the number of components c and the size t of the tree T' . In some cases, the parametrization does

		PINT	INT	PATH	CHOR
FIXED	RECOG*	$\mathcal{O}(n + m)$	$\mathcal{O}(n + m)$	NP-complete	NP-complete
	REPEXT	NP-complete	NP-complete	NP-complete	NP-complete
SUB	RECOG*	$\mathcal{O}(n + m)$ [15, 3]	$\mathcal{O}(n + m)$ [2, 4]	NP-complete	NP-complete
	REPEXT	$\mathcal{O}(n + m)$	$\mathcal{O}(n + m)$	NP-complete	NP-complete
ADD	RECOG*	$\mathcal{O}(n + m)$ [15, 3]	$\mathcal{O}(n + m)$ [2, 4]	$\mathcal{O}(nm)$ [8, 17]	$\mathcal{O}(n + m)$ [16]
	REPEXT	$\mathcal{O}(n + m)$	NP-complete	NP-complete	NP-complete
BOTH	RECOG*	$\mathcal{O}(n + m)$ [15, 3]	$\mathcal{O}(n + m)$ [2, 4]	$\mathcal{O}(nm)$ [8, 17]	$\mathcal{O}(n + m)$ [16]
	REPEXT	$\mathcal{O}(n + m)$ [13]	$\mathcal{O}(n + m)$ [1]	open	NP-complete

Fig. 2. Table of the complexity of different problems for all four considered classes. Results without references are new results of this paper.

not help and the problem is **NP**-complete even if the value of the parameter is zero or one. In other cases, the problems are fixed-parameter tractable (FPT), $W[1]$ -hard or in XP.

The main result concerning parametrization is the following. The BINPACKING problem is a well-known problem concerning integer partitions; more details in Section 3. For two problems A and B , we denote by $A \leq B$ a polynomial reduction and by $A \leq_{\text{wtt}} B$ a weak truth-table reduction (roughly, we may use a number of B -oracle questions bounded by a computable function to solve A):

Theorem 1. $\text{BINPACKING} \leq \text{REPEXT}(\text{PINT}, \text{FIXED}) \leq_{\text{wtt}} \text{BINPACKING}$ where the weak truth-table reduction needs to solve 2^k instances of BINPACKING.

1.4 Related Problems

Last, we describe two problems which are closely related to our results. The problem of simultaneous representations [10] asks whether there exist representations $\mathcal{R}_1, \dots, \mathcal{R}_k$ of graphs G_1, \dots, G_k which are the same on the common part of the vertex set $I = V(G_i) \cap V(G_j)$ for all $i \neq j$. It was noted in [14] that the partial representation extension is closely related to the simultaneous representations. For instance, using simultaneous representations of interval graphs, we can solve their partial representation extension [1]. This is not the case for chordal graphs since REPEXT of chordal graphs is **NP**-complete but their simultaneous representations are solvable in polynomial-time.

The partial representation extension of proper interval graphs described here is closely related to partial representations and bounded representations of unit interval graphs [13]. In both cases we deal with interval representations in a limited space. So the techniques initially developed for unit interval graphs are easily used here for proper interval graphs. We note that the problems concerning unit interval graphs are more difficult since they involve computations with rational number positions.

2 Preliminaries

In this section, we describe notation used in the paper. Also, we deal with two common concepts: located and unlocated components, and groups of indistinguishable vertices.

Notation. We consider finite undirected simple graphs, i.e., graphs without loops or multiple edges. As usual, we reserve n for the number of vertices and m for the number of edges of the main considered graph G . The set of vertices is denoted by $V(G)$ and the set of edges by $E(G)$. For a vertex $v \in V(G)$, we let $N(v) = \{x : vx \in E(G)\}$ denote the open neighborhood of v , and $N[v] = N(v) \cup \{v\}$ the closed neighborhood of v .

Let P_n denote the path of length n (with $n + 1$ vertices). For a tree, we call the vertices of degree larger than two *branch vertices* and the vertices of degree at most two *non-branch vertices*, and of course vertices of degree one *leaves*.

Types of Components. For the partial representation extension problem, the graph G contains two types of components. A component C is called a *located component* if it has at least one vertex pre-drawn, i.e., $C \cap G'$ is non-empty. A component C is called an *unlocated component* if no interval is pre-drawn, i.e., $C \cap G' = \emptyset$.

In the case of interval graphs, if the partial representation is extendible, the located components has to be ordered from left to right. This ordering is given by the ordering of the pre-drawn intervals from left to right. If for some $u \in C$ and $v, w \in C'$, $C \neq C'$, is R_u between R_v and R_w , the ordering does not exist. But in such a case the partial representation is clearly not extendible. Descriptions of the algorithms ignore these trivial cases.

Indistinguishable Vertices. Let u and v be two vertices of G such that $N[u] = N[v]$. These two vertices are called *indistinguishable* since they can be represented exactly the same, having $R_u = R_v$ (a common property of all intersection representations). From the structural point of view, *groups of indistinguishable* vertices are not very interesting. The goal is to construct a pruned graph where each group is represented by a single vertex. For that, we need to be little careful since we cannot prune pre-drawn vertices.

For an arbitrary graph, its groups of indistinguishable vertices can be located in time $\mathcal{O}(n + m)$ [16]. We prune the graph in the following way. If u and v are indistinguishable and u is not pre-drawn, we eliminate it from the graph (and for the representation, we can put $R_u = R_v$). The resulting pruned graph has the following property: If two vertices u and v indistinguishable, they are both pre-drawn. For the rest of the paper, we expect that all input graphs are pruned.

Maximal Cliques. There is the following property of maximal cliques, valid for all representations by subtrees inside a tree. Let $x \in T$ and consider the set $K = \{u : u \in V(G), x \in R_u\}$. Clearly, K is a clique of G .

On the other, let K be a maximal clique of G . Subtrees of tree have the Helly property which states that every pairwise intersecting collection of subtrees have a common intersection as a subtree. Since the subtrees representing K are pairwise intersecting, the common intersection $R_K = \cap_{u \in K} R_u$ is a subtree of T . This subtree R_K is not intersected by any other R_v for $v \notin K$ (otherwise K would not be a maximal clique). Thus the subtrees R_K corresponding to different maximal cliques are pairwise disjoint. For example, if $|T|$ is smaller than the number of maximal cliques of G , the graph is clearly not representable in T .

3 Interval Graphs

In this section, we deal with classes PINT and INT. The results obtained here are used as tools for PATH and CHOR graphs in Section 4.

3.1 Polynomial Cases

First we deal with all polynomial cases. Also, we describe several concepts as minimum span, useful in the rest of the paper.

Non-fixed Type Recognition. The only limitation for recognition of interval graphs inside a given path is the length of the path. In all three types SUB, ADD and BOTH, we can produce a path as long as necessary. For an interval representation, only the order of the endpoints from left to right is important, not the exact positions. In a tree T with at least $2n$ vertices, every possible ordering is realizable.

Thus the problems are equivalent to the standard recognition on the real line. For PINT, it can be solved in time $\mathcal{O}(n + m)$, for example [15, 3]. Similarly for INT, it can be solved in linear-time [2, 4].

Both Type Extension. This extension type is equivalent with the partial representation extension problems on the real line. Again only the ordering of the endpoints is important. The only change here is that some of the endpoints are already placed. By subdividing, we can place any amount of the endpoints between any two endpoints (not sharing the same position). Also, the path can be extended to the left and to the right which allows to place any amount of endpoints to the left of the left-most pre-drawn endpoint and to the right of the right-most pre-drawn endpoint. So any extending ordering can be realized in the BOTH type.

The partial representation extension problem for interval graphs on the real line was first considered in [14]. The paper gives algorithms for both classes INT and PINT and does not explicitly deal with representations sharing endpoints but the algorithms are easy to modify. New results [13, 1] show that the both extension problems are solvable in time $\mathcal{O}(n + m)$.

Sub Type Extension. It is possible to modify the above algorithms for partial representation extension of INT and PINT. Since we do not want to go in details of these algorithms, we instead reduce to BOTH type extensions which we can solve in time $\mathcal{O}(n + m)$ (as discussed above):

Theorem 2. *The both problems REPEXT(PINT, SUB) and REPEXT(INT, SUB) can be solved in time $\mathcal{O}(n + m)$.*

Proof. First, we describe the algorithm for INT. Let p_1, \dots, p_t be the vertices of the path T' . If the graph contains unlocated components, we first deal with them. They can be placed if one of the following works, otherwise the representation is not extendible.

- If there is only a single located component, then at least one of p_1 and p_t has to be not contained in any pre-drawn interval. If so, we can subdivide the end of the path T' and place all the unlocated components there. Otherwise, the unlocated components cannot be placed at all.
- If there are more located components, they have to be ordered correctly from left to right $C_1 < \dots < C_c$. Now, we can subdivide an edge between these components and place the unlocated components there.

What remains is to deal with the located components.

If at least one of p_1 and p_t is contained in some pre-drawn interval, we modify both the path and the graphs. If $v_1, \dots, v_k \in C_1$ such that $p_1 \in R_{v_1}, \dots, R_{v_k}$, we modify as follows. First, we extend the path by one by adding p_0 attached to p_1 . We introduce an additional pre-drawn interval v_{\leftarrow} adjacent exactly to v_1, \dots, v_k with $R_{v_{\leftarrow}} = \{p_0\}$ and we modify $R'_{v_i} = R_{v_i} \cup \{p_0\}$. Exactly the same modification introducing p_{t+1} and v_{\rightarrow} is used for the right-end if there is some $v \in C_c$ such that $p_t \in R_v$.

We use the described algorithm for REPEXT(INT, BOTH) for the modified graph and the modified path, which runs in time $\mathcal{O}(n + m)$. What remains is to argue correctness, which we do only for left end of T ; for the right end the argument is symmetric. If nothing pre-drawn contains p_1 , the edge $p_1 p_2$ can be subdivided as necessary, thus creating the exactly same situation as in BOTH type. On the other hand, if p_0 was added, everything has to be represented on the right of v_{\leftarrow} which is placed on p_0 .

But there is a single problem we need to deal with. The newly added edge $p_0 p_1$ can be subdivided. There are two possibilities. If $|R_{v_i}| \geq 2$ for each i , the subdivision of $p_0 p_1$ is equivalent to subdivision of $p_1 p_2$ which is correct in the original problem. Now let $|R_{v_i}| = 1$ for some i . Then $N(v_i)$ has to be a complete subgraph. In such a case, we can revert the subdivision of $p_0 p_1$ as follows. Let p'_1, \dots, p'_s be the newly created vertices by subdividing $p_0 p_1$. For each $v \in N(v_i)$ (of course, with exception of v_{\leftarrow}), we put $R'_v = R_v \setminus \{p'_1, \dots, p'_s\} \cup \{p_1\}$, and we remove p'_1, \dots, p'_s by contractions.

Now, by removing $p_0, p_{t+1}, v_{\leftarrow}$ and v_{\rightarrow} (of course, only if they are added), we obtain a correct representation of G inside a subdivision of T' extending the partial representation. Concerning proper interval graphs, we use almost the same approach. One change is that we append two vertices \bar{p}_0 and p_0 (resp. p_{t+1} and \bar{p}_{t+1}) to the end of T' and put $R_{v_{\leftarrow}} = \{\bar{p}_0, p_0\}$ (resp. $R_{v_{\rightarrow}} = \{p_{t+1}, \bar{p}_{t+1}\}$), so the resulting partial representation is proper. Concerning the case $|R_{v_i}| = 1$ for some i , it is trivial since $k = 1$ and $N(v_i)$ is empty, and the component C_1 has to be the single interval. \square

General Properties of PINT. For each representation, we have some ordering $<$ of intervals from left to right. This is an ordering of left-endpoints from left to right (and at the same time the ordering of the right-endpoints). The following lemma states uniqueness of this ordering [5]:

Lemma 1 (Deng et al.). *For a connected proper interval graph, the ordering $<$ is uniquely determined up to local reordering of groups and complete reversal.*

Now let us consider the types FIXED and ADD. The space on the path is limited and it is important to know how much space does a component C require. We denote it by $\text{minspan}(C)$. Let \mathcal{R} be some representation of C , p_i be the left-most vertex of T' contained in representation of C and p_j be the right-most such vertex. Then

$$\text{minspan}(C) = \begin{cases} \min_{\mathcal{R}} \{j - i + 1\} & \text{if some representation of } C \text{ exists,} \\ +\infty & \text{otherwise.} \end{cases}$$



Fig. 3. The ordering \triangleleft is $\ell_1 \triangleleft \ell_2 \triangleleft r_1 \triangleleft \ell_3 \triangleleft \ell_4 \triangleleft r_2 \triangleleft r_3 \triangleleft \ell_5 \triangleleft r_4 \triangleleft \ell_6 \triangleleft r_5 \triangleleft r_6$ for the component on the left. The constructed smallest possible representation of the component C on the right, with $\text{minspan}(C) = 8$.

Lemma 2. *For every component C (both located, or unlocated), the value $\text{minspan}(C)$ can be computed in $\mathcal{O}(n + m)$ (together with a realizing representation).*

Proof. (1) First, we deal with an unlocated component. We start by constructing any ordering $<$ of the left-endpoints of intervals from left to right from Lemma 1, in time $\mathcal{O}(n + m)$ using the algorithm of Corneil et al. [3]. Using this ordering, we want to produce a representation as small as possible.

Let ℓ_i denote the left-endpoint and r_i the right-endpoint of the interval v_i . From ordering $v_1 < \dots < v_n$ and $\ell_1 < \dots < \ell_n$, we want to compute a common ordering \triangleleft of both the left-endpoints and the right-endpoints. The starting point is an ordering of just the left-endpoints $\ell_1 \triangleleft \dots \triangleleft \ell_n$. Into this ordering, we insert right-endpoints r_1, \dots, r_n one-by-one. A right-endpoint r_i is inserted right before ℓ_j where v_j is the left-most non-neighbor of v_i on the right (or if such v_j does not exist, we append r_i to the end). By Lemma 1, the ordering \triangleleft is uniquely defined since $<$ is unique. For an example of construction of \triangleleft , see Figure 3.

Now, we construct the smallest representation as follows. Let p_1, \dots, p_k be the vertices of the tree T . We construct an assignment f which maps endpoints into T . Then for a vertex v_i we put $R_{v_i} = \{p_j : f(\ell_i) \leq p_j \leq f(r_i)\}$. The mapping f is constructed for endpoints one-by-one, according to \triangleleft . If the previous endpoint in \triangleleft had assigned vertex p_i , we assign to the current endpoint: If the current endpoint is a right-endpoint and the previous endpoint is a left-endpoint, assign p_i . Otherwise assign p_{i+1} . In total, the component needs $2n - \ell$ vertices of T where ℓ denotes the number of changes from a left-endpoint to a right-endpoint in the ordering \triangleleft ; so $2n - \ell$ is the value of $\text{minspan}(C)$. For an example, see Figure 3. The total complexity of the algorithm is clearly $\mathcal{O}(n + m)$.

To conclude the proof, we need to show that the constructed representation is correct and the smallest one. If $v_i v_j \in E$ and $v_i < v_j$, then $\ell_i \triangleleft \ell_j \triangleleft r_i$ and R_{v_i} intersects R_{v_j} (between $f(\ell_j)$ and $f(r_i)$). If $v_i v_j \notin E$, then $r_i \triangleleft \ell_j$; a property of $<$ is that the neighborhood of every vertex is consecutive in $<$. Thus r_i is placed on the left of ℓ_j in \triangleleft . Therefore, $R_{v_i} \cap R_{v_j} = \emptyset$ as required.

Concerning minimality notice that in a pruned graph it always holds $\ell_i \neq \ell_j$ and $r_i \neq r_j$ for every pair v_i and v_j . We argue that we use gaps as small as possible. Only a right-endpoint r_i following a left-endpoint ℓ_j can be placed at the same position. The other case of a right-endpoint r_i followed by a left-endpoint ℓ_j requires a gap of size one; otherwise R_{v_i} would intersect R_{v_j} but $v_i v_j \notin E$. So the gaps are minimal and we construct a smallest representation and give the value $\text{minspan}(C)$ correctly.

(2) Concerning a located component, we just modify the above approach slightly. The ordering $<$ has to be compatible with the ordering of the pre-drawn intervals. For a group of indistinguishable intervals, all its intervals are pre-drawn and we know their ordering from left to right. So we just test whether $<$ and its reversal can be used. We have (at most) two possibilities for $<$ which give the same $\text{minspan}(C)$ but the minimum representations might be differently shifted, and we are able to construct both of them. If the pre-drawn intervals do not belong to one group, the ordering $<$ is uniquely defined (if it is compatible with the ordering of pre-drawn intervals at all).

We compute the common ordering \triangleleft as before and place the endpoints in this ordering. The only exception is that the endpoints of the pre-drawn intervals are fixed. Additionally, we place the endpoints on the left of the left-most pre-drawn endpoint as far to the right as possible and similarly as far to the left as possible on the right of the right-most pre-drawn endpoint. The constructed representation is the smallest possible and gives $\text{minspan}(C)$. \square

General Properties of INT. Recall properties of maximal cliques from Section 2. For a component C , we denote by $\text{cl}(C)$ the number of maximal cliques of C . Since the subtrees R_K corresponding to the maximal cliques are disjoint, they have to be ordered from left to right. There is the following well-known property of this ordering [6]:

Lemma 3 (Fulkerson and Gross). *A graph is an interval graph if and only if there exists an ordering of the maximal cliques $K_1 < \dots < K_{\text{cl}(C)}$ such that for every vertex the cliques containing this vertex appear consecutively in this ordering.*

We quickly argue correctness of the lemma. Clearly, in an interval representation, all maximal cliques corresponding to one vertex v are appearing consecutively (otherwise the clique in between would be intersected by R_v in addition). On the other hand, having an ordering $<$ of the maximal cliques from the statement, we can construct a representation as follows. To each clique K_i , assign one vertex p_i of T . Now for each vertex v , we assign $R_v = \{p_i : v \in K_i\}$. Since the maximal cliques appear consecutively, we assign a subpath to each interval. Also, the representation is correct.

For types FIXED and ADD, we again consider minimum span defined exactly as for proper interval graphs above. Clearly, $\text{minspan}(C) \geq \text{cl}(C)$. We show:

Lemma 4. *For an unlocated component C , $\text{minspan}(C) = \text{cl}(C)$ if C is a component of an interval graph. We can find a smallest representation in time $\mathcal{O}(n + m)$.*

Proof. We start by identifying maximal cliques in time $\mathcal{O}(n + m)$, using algorithm of Rose et al. [16]. To construct a smallest representation, we find an ordering from Lemma 3, using the PQ-tree algorithm [2] in time $\mathcal{O}(n + m)$. If such an ordering does not exist, the graph G is not an interval graph and no representation exists. If the ordering exists, we can construct a representation using exactly $\text{cl}(C)$ vertices of the path as described above. \square

Fixed Type Recognition. We just need to use the values minspan we already know how to compute.

Proposition 1. *Both $\text{RECOG}^*(\text{PINT}, \text{FIXED})$ and $\text{RECOG}^*(\text{INT}, \text{FIXED})$ can be solved in time $\mathcal{O}(n + m)$.*

Proof. We process components C_1, \dots, C_c one-by-one and place them from left to right on T' . If $\sum_{i=1}^c \text{minspan}(C_i) \leq |T'|$, we can place the components using the smallest representation from Lemma 2 for **PINT**, resp. Lemma 4 for **INT**. Otherwise, the path is too small and the representation cannot be constructed. \square

Add Type Extension, PINT. Again, we approach this problem using minimum spans and Lemma 2.

Theorem 3. *The problem $\text{REPEXT}(\text{PINT}, \text{ADD})$ can be solved in time $\mathcal{O}(n+m)$.*

Proof. Since the path can be expanded to left and to right as much as necessary, we can place unlocated components far to the left. So we only need to deal with located components, ordered $C_1 < \dots < C_c$. We place components from left to right. When we place C_i , it has to be placed on the right of C_{i-1} . We have (at most) two possible smallest representations corresponding to two different orderings of C_i . We test whether at least one of them can be placed on the right of C_{i-1} and pick the one minimizing the right-most vertex taken by C_i (leaving maximum possible space for C_{i+1}, \dots, C_c). If neither representation can be placed, the extension algorithm fails.

Now, if the algorithm finishes, it constructs a correct representation. On the other hand, we place each component as far to the left as possible (while restricted by the previous components on the left). So if C_i cannot be placed, there exists no representation extending the partial representation. \square

3.2 NP-complete Cases

The basic gadgets of the reduction are paths. They have the following minimum spans.

Lemma 5. *For **INT**, $\text{minspan}(P_n) = n$. For **PINT** and $n \geq 2$, $\text{minspan}(P_n) = n + 2$.*

Proof. For **INT**, the number of maximal cliques of P_n is n . For **PINT**, the ordering \triangleleft is

$$\ell_0 \triangleleft \ell_1 \triangleleft r_0 \triangleleft \ell_2 \triangleleft r_1 \triangleleft \dots \triangleleft \ell_i \triangleleft r_{i-1} \triangleleft \dots \triangleleft \ell_n \triangleleft r_{n-1} \triangleleft r_n.$$

So there are n changes from ℓ_i to r_{i-1} and the minimum span is $n + 2$. \square

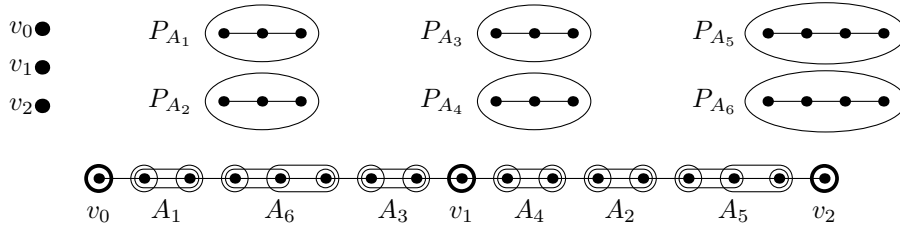


Fig. 4. An example of the reduction for the input set of 3-PARTITION: $k = 2$, $M = 7$, $A_1 = A_2 = A_3 = A_4 = 2$ and $A_5 = A_6 = 3$. On top, the constructed interval graph is depicted. On bottom, the partial representation (depicted in bold) is extended.

We reduce the problem from 3-PARTITION. The input of 3-PARTITION consists of integers k , M and A_1, \dots, A_{3k} such that $\frac{M}{4} < A_i < \frac{M}{2}$ for each A_i and $\sum A_i = kM$. It asks whether it is possible to partition A_i 's into k triples such that the sets A_i of each triple sum to exactly M .² This problem is known to be strongly NP-complete (even with all integers of a polynomial size) [7].

Theorem 4. *The problems REPEXT(PINT, FIXED) and REPEXT(INT, FIXED) are NP-complete.*

Proof. The described reduction works for both PINT and INT (with a small modification). For a given input of 3-PARTITION (with $M \geq 4$), we construct a graph G and its partial representation. As the fixed tree we choose $T' = P_{(M+1)k}$, with the vertices $p_0, \dots, p_{(M+1)k}$.

Now, the graph G contains two types of gadgets as separate components. First, it contains $k + 1$ *split gadgets* S_0, \dots, S_k which split the path into k gaps of equal sizes M . Then it contains $3k$ *take gadgets* T_1, \dots, T_{3k} . A take gadget T_i takes in each representation space at least A_i of one of the k gaps.

For this reduction, the gadgets are particularly simple. The split gadget S_i is just a single pre-drawn vertex v_i with $R_{v_i} = \{p_{(M+1)i}\}$. The split gadgets clearly split the path into k equal gaps of size M . The take gadget T_i is a P_{A_i} for INT, resp. P_{A_i-2} for PINT. According to Lemma 5, $\text{minspan}(T_i) = A_i$. The representation is extendible if and only if it is possible to place the take gadgets into the k gaps. For an illustration, see Figure 4. The reduction is clearly polynomial.

To conclude the proof, we show that the partial representation is extendible if and only if the corresponding 3-PARTITION input has a solution. If the partial representation is extendible, the take gadgets T_i are divided into the k gaps on the path which forms a partition. Based on conditions for sizes of A_i 's, each gap contains exactly three take gadgets of the total minimum span M ; thus the partition solves the 3-PARTITION problem. On the other hand, a solution of 3-PARTITION describes how to place the take gadgets into the k gaps and construct the extending representation. \square

² Notice that if a subset of A_i 's sums to exactly M it has to be a triple due to size conditions.

Corollary 1. *The problem $\text{REPEXT}(\text{INT}, \text{ADD})$ is NP-complete.*

Proof. Use the above reduction with one additional pre-drawn interval v attached to everything in G . We put $R_v = \{p_0, \dots, p_{(M+1)k}\}$, so it contains the whole tree T' . Now since representation of each T_i has to intersect R_v , it has to be placed inside of the k gaps as before. \square

3.3 Parametrized Complexity

In this subsection, we study parametrized complexity. The parameters are the number of components c , the number of pre-drawn intervals k and the size of the tree t .

By Number of Components. In the above reduction, one might ask whether it is possible to make the reduction graph G connected. For INT , it is indeed possible to add universal vertices adjacent to everything in G and thus making G connected, as in the proof of Corollary 1. The following results shows that for PINT it is not possible (unless $\text{P} = \text{NP}$):

Proposition 2. *The problem $\text{REPEXT}(\text{PINT}, \text{FIXED})$ is fixed-parameter tractable in the number of components c , solvable in time $\mathcal{O}((n+m)c!)$.*

Proof. There is $c!$ possible orderings of the components from left to right, we test each (some ordering might be forced by pre-drawn intervals). We show that for a given ordering of components, we can solve the problem in time $\mathcal{O}(n+m)$; thus gaining the total time $\mathcal{O}((n+m)c!)$. We solve the problem almost the same as in the proof of Theorem 3. The only difference is that we deal with all the components instead of only the located components.

We process the components from left to right. When we process C_i , we place it on the right of C_{i-1} as far to the left as possible. For unlocated C_i , we can take any ordering. For located C_i , we test both orderings and take the one placing C_i further to the left. We construct the representation in time $\mathcal{O}(n+m)$ and the algorithm is clearly correct; see the proof of Theorem 3 for more details. \square

By Number of Pre-drawn Intervals. In the reduction in Theorem 4, we need to have k pre-drawn intervals. One could ask, whether the problems becomes simpler with a small number of pre-drawn intervals. We answer this negatively, for PINT the problem is in XP and $\text{W}[1]$ -hard with respect to k .

We start with two closely related problems BINPACKING and GENBINPACKING . In the both problems, we have k bins and n items of integer sizes. The question is whether we can pack (partition) these items into the k bins when sizes of the bins are limited. For BINPACKING , all the bins have the same size. For GENBINPACKING , the bins have different sizes. Formally:

Problem: BINPACKING

Input: Integers k, ℓ, V and A_1, \dots, A_ℓ .

Output: Does there exist a k -partition $\mathcal{P}_1, \dots, \mathcal{P}_k$ of A_1, \dots, A_ℓ such that $\sum_{A_i \in \mathcal{P}_j} A_i \leq V$ for every \mathcal{P}_j .

Problem: GENBINPACKING

Input: Integers k, ℓ, V_1, \dots, V_k and A_1, \dots, A_ℓ .

Output: Does there exist a k -partition $\mathcal{P}_1, \dots, \mathcal{P}_k$ of A_1, \dots, A_ℓ such that $\sum_{A_i \in \mathcal{P}_j} A_i \leq V_j$ for every \mathcal{P}_j .

Lemma 6. *The problems BINPACKING and GENBINPACKING are polynomially equivalent.*

Proof. Obviously BINPACKING is a special case of GENBINPACKING. On the other hand, let k, ℓ, V_1, \dots, V_k and A_1, \dots, A_ℓ be an instance of GENBINPACKING. We construct an instance k', ℓ', V' and $A'_1, \dots, A'_{\ell'}$ of BINPACKING as follows. We put $k' = k, \ell' = \ell + k$ and $V' = 2 \max V_i + 1$. The weights of the first ℓ items are the same, i.e. $A'_i = A_i$ for $i = 1, \dots, \ell$. The added items $A'_{\ell+1}, \dots, A'_{\ell+k}$ are called *large* and we put $A'_{\ell+i} = V' - V_i$ for $i = 1, \dots, k$.

Now, each bin has to contain exactly one large item since two large items take more than V' . After placing large items into the bins, we obtain bins of sizes V_1, \dots, V_k in which we have to place the remaining items. This corresponds exactly to the original GENBINPACKING instance. \square

If the input sizes are encoded in binary, the problem is NP-complete even for $k = 2$. The more interesting version which we use for the rest of the paper is that the sizes are encoded in unary so all sizes are polynomial. In such a case, the BINPACKING problem is known to be solvable in time $t^{\mathcal{O}(k)}$ using dynamic programming where t is the total size of all items. And it is W[1]-hard with respect to the parameter k [11]. Similar holds for REPEXT(PINT, FIXED):

Proof (Theorem 1). For a given instance of the BINPACKING problem, we can solve it by REPEXT(PINT, FIXED) in a similar manner as in the reduction in Theorem 4. As T' , take a path $P_{(V+1)k}$. As G , take P_{A_i-2} for each A_i and pre-drawn vertices v_0, \dots, v_k such that $R_{v_i} = \{p_{(V+1)i}\}$. The rest of the argument is exactly as in the proof of Theorem 4.

Now, we want to solve REPEXT(PINT, FIXED) using 2^k instances of GENBINPACKING (which is polynomially equivalent to BINPACKING), where k is the number of pre-drawn intervals.

First we deal with located components $C_1 < \dots < C_c$. For each component, we have two possible orderings $<$ and using Lemma 2 we get (at most) two possible smallest representations which might be differently shifted. In total, we have at most $2^c \leq 2^k$ possible representations keeping C_1, \dots, C_c as small as possible leaving maximum possible gaps for unlocated components. We test each of these representations.

Now, let $C'_1, \dots, C'_{c'}$ be the unlocated components. For each C'_i , we compute $\text{minspan}(C'_i)$ using Lemma 2. The goal is to place unlocated components in the $c + 1$ gaps between representations of the located components C_1, \dots, C_c . We can solve this problem using GENBINPACKING as follows. We have $k + 1$ bins of sizes equal to the gaps between the representations of C_1, \dots, C_c . We have c' items of sizes $A_i = \text{minspan}(C'_i)$. The solution of GENBINPACKING tells in

which gaps the unlocated components can be placed. If there exists no solution, this specific smallest representation of located components can not be used.

We can test all 2^k possible representations. Thus we get the required weak truth-table reduction. \square

Corollary 2. *The problem $\text{REPEXT}(\text{PINT}, \text{FIXED})$ is $\text{W}[1]$ -hard and belongs to XP , solvable in time $n^{\mathcal{O}(k)}$ where k is the number of pre-drawn intervals.*

Proof. Both obtained easily by Theorem 1. \square

Proposition 3. *The problems $\text{REPEXT}(\text{INT}, \text{FIXED})$ and $\text{REPEXT}(\text{INT}, \text{ADD})$ are $\text{W}[1]$ -hard with respect to the parameter k where k is the number of pre-drawn intervals.*

Proof. Modify the reduction in Theorem 4 and Corollary 1 as in the proof of Theorem 1. \square

By Size of the Path. We show that the NP-complete cases are fixed-parameter tractable with respect to the size of the path t . It is easy to find a solution by a brute-force algorithm:

Proposition 4. *For t the size of T' , the problems $\text{REPEXT}(\text{PINT}, \text{FIXED})$ and $\text{REPEXT}(\text{INT}, \text{FIXED})$ are fixed-parameter tractable with respect to the parameter t . They can be solved in time $\mathcal{O}(n + m + f(t))$ where $f(t) = t^{2t^2}$.*

Proof. In the pruned graph, the non-pre-drawn vertices has to be represented pairwise different. There are at most t^2 possible different subpaths of a path with t vertices so the pruned graph can contain at most t^2 different vertices; otherwise the extension is not possible. We can test every possible assignment of t^2 non-pre-drawn vertices to t^2 subpaths, and for each assignment we test whether it is a correct representation. \square

4 Path and Chordal graphs

We present several results concerning PATH and CHOR classes. We use many results from Section 3 as basic tools here.

4.1 Polynomial Cases

The recognition problem for types ADD and BOTH is equivalent to standard recognition without any additional tree T' . Indeed, we can modify T' by adding an arbitrary tree to it. If the input graph is either PATH or CHOR , there exists a tree T'' in which the graph can be represented. We produce T by attaching T'' to T' in any way. Clearly, the graph can be represented in T as well, completely ignoring the part T' for example.

For path graphs, the original recognition algorithm is due to Gavril [8] in time $\mathcal{O}(n^4)$. The current fastest algorithm is by Schaffer [17] in time $\mathcal{O}(nm)$. For chordal graphs, there is a beautiful simple algorithm in time $\mathcal{O}(n + m)$ by Rose et al. [16].

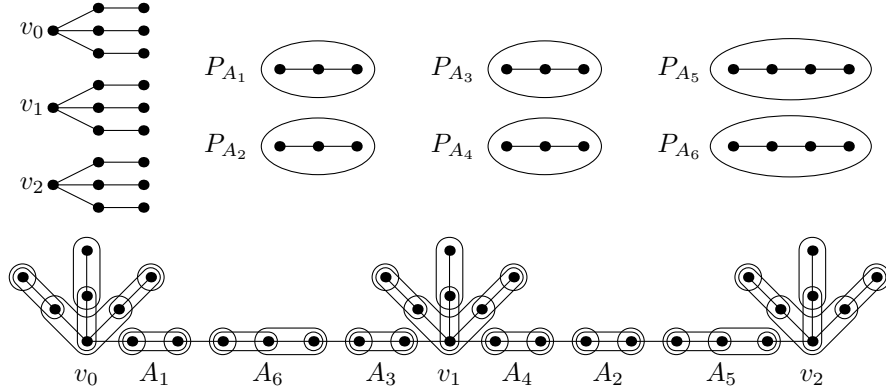


Fig. 5. For the same input set of 3-PARTITION, on top the constructed graph is depicted. On bottom, a representation is constructed, giving a solution $\{A_1, A_3, A_6\}$ and $\{A_2, A_4, A_5\}$.

4.2 NP-complete Cases

We describe a modification of the reduction of Theorem 4 for path and chordal graphs. We start with the simplest reduction for the FIXED type and then modify it for other problems.

Fixed Type. For FIXED type, we can avoid pre-drawn subtrees, using an additional structure of the tree.

Proposition 5. *Both $\text{RECOG}^*(\text{PATH}, \text{FIXED})$ and $\text{RECOG}^*(\text{CHOR}, \text{FIXED})$ are NP-complete.*

Proof. We again reduce from 3-PARTITION with an input k and M , and for technical purposes let $M \geq 4$. We construct a graph G and a tree T as follows. The tree T is a path $P_{(M+1)k}$ (denote its vertices $p_0, \dots, p_{(M+1)k}$) with three paths of length two attached to every vertex $p_{(M+1)i}$, for each $i = 0, \dots, k$; see Figure 5.

Each split gadget S_i is a star, depicted on the left of Figure 5. When the split gadgets are placed as in the figure, they split the tree into k gaps exactly as the pre-drawn vertices in the proof of Theorem 4. Each take gadget T_i is the same path P_{A_i} as before.

What remains is to argue correctness of the reduction. We claim that each R_{v_i} contains at least one branch vertex (actually exactly one, since there are exactly $n + 1$ branch vertices in T). If some R_{v_i} would only contain non-branch vertices, then it would not be possible to represent three disjoint neighbors u_1 , u_2 and u_3 of v_i ; each $R_{u_j} \setminus R_{v_i}$ has to be non-empty.

Now since each branch vertex is taken by one R_{v_i} , the tree T is split into k gaps as before. Since $|A_i| > 2$, each T_i can be represented only inside of these gaps. Notice that the total sum of free vertices in the gaps has to be kM , and

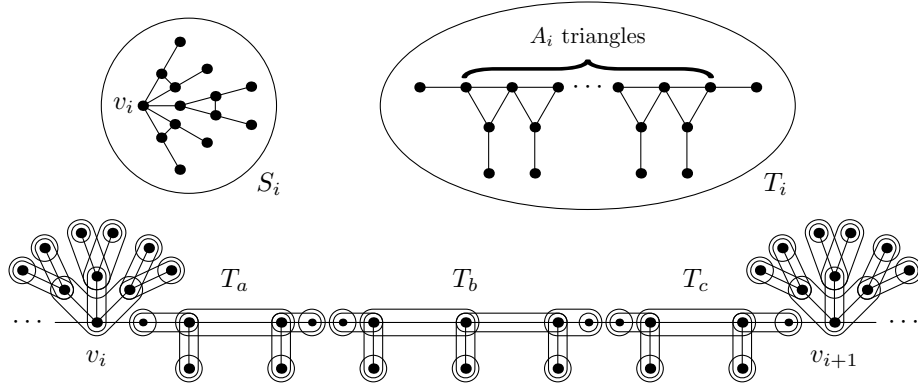


Fig. 6. On top, the split gadget S_i on left and the take gadget T_i on right. On bottom, a part of the tree T , the small vertices are added by subdivision. The gap between two split gadgets contains three take gadgets T_a , T_b and T_c giving one triple $\{A_a, A_b, A_c\}$ with $A_a + A_b + A_c = M$.

therefore the split gadgets has to be represented entirely in the attached stars as in Figure 5. The rest of the reduction works exactly as in Theorem 4. \square

Sub Type. By modifying the above reduction, we get:

Theorem 5. *The problems $\text{RECOG}^*(\text{PATH}, \text{SUB})$ and $\text{RECOG}^*(\text{CHOR}, \text{SUB})$ are NP-complete.*

Proof. We need to modify the two gadgets from the reduction in Theorem 5 in such a way that subdivision of the tree does not help in placing them. Subdivision only increases the number of non-branch vertices. Thus a take gadget T_i requires A_i branch vertices. Similarly, the split gadget S_i is more complicated. See Figure 6 on top.

The tree T is constructed as follows. We start with a path $P_{(M+1)k}$ with vertices $p_0, \dots, p_{(M+1)k}$. To each vertex $p_{(M+1)i}$ we attach a subtree isomorphic to the trees in Figure 6 on bottom. To the other vertices of the path, we attach one leaf per a vertex.

Clearly, for a given solution of 3-PARTITION, we can construct a correct representation in a subdivided tree. On the other hand, we are going to show how to construct a solution of 3-PARTITION from a given tree representation.

Recall maximal cliques from Section 2. Notice that each triangle $u_1u_2u_3$ in the gadgets is a maximal cliques K (and for each triangle, we get a different maximal clique). As such, R_K has to contain a branch vertex since $N[u_i] \neq N[u_j]$ for each $i \neq j$. The gadget S_i contains three triangles, each taking one branch vertex of T . In addition, their R_K 's are connected by R_{v_i} which has to contain another branch vertex. So in total, S_i contains at least four branch vertices. The gadget T_i contains A_i triangles, and so it requires at least A_i branch vertices. Since

the number of branch vertices of T is limited, each S_i takes exactly four branch vertices and each T_i takes exactly A_i branch vertices.

Now, if some T_i would contain a branch vertex of the subtrees attached to $p_{(M+1)j}$, at least on its branch vertices would not be used (either not taken by T_i or T_i would require at least $A_i + 1$ branch vertices). So each S_i have to take branch vertices of the subtrees attached to $p_{(M+1)j}$ for some j , and the take gadgets have to be placed inside the gaps exactly as before. \square

Theorem 6. *Even with only a single subtree pre-drawn, i.e., $|G'| = 1$, the problems $\text{REPEXT}(\text{CHOR}, \text{ADD})$ and $\text{REPEXT}(\text{CHOR}, \text{BOTH})$ are NP-complete.*

Proof. We easily modify the above reductions; for ADD type, the reduction of Proposition 5, for BOTH type, the reduction of Theorem 5. The modification adds one pre-drawn vertex v into G adjacent to everything such that $R_v = T'$. The representation of v spans the whole tree and thus forces the entire representation into T' .

We now deal just with ADD type, for BOTH is the argument exactly the same. Let T' be the partial tree and let T be the tree in which the representation is constructed, so T' is a subtree of T . We claim that we can restrict a representation of each vertex of G into T' and thus obtain a correct representation inside the subtree T' .

Let $x \in V$. Since $xv \in E$, the intersection of R_x and T' is a non-empty subtree. So $R'_x = R_x \cap T'$ is a representation of G in T' . To argue the correctness, let x and y be two different vertices from v (otherwise trivial). If $xy \notin E$, then $R_x \cap R_y = \emptyset$, and so even $R'_x \cap R'_y = \emptyset$. Otherwise xyv is a triangle in G and thus by the Helly property the subtrees R_x , R_y and $R_v = T'$ have a non-empty common, giving $R'_x \cap R'_y$ non-empty. Thus, this gives a reduction from $\text{REPEXT}(\text{CHOR}, \text{FIXED})$. \square

For path graphs, one can use a similar technique of a pre-drawn universal vertex attached to everything. But there is the following difficulty: To do so, the input partial tree T' has to be a path. For type BOTH, the complexity of $\text{REPEXT}(\text{PATH}, \text{BOTH})$ remains open. For type ADD, we get the following weaker result:

Proposition 6. *The problem $\text{REPEXT}(\text{PATH}, \text{ADD})$ is NP-complete.*

Proof. Similarly as in Theorem 6, add a pre-drawn universal vertex v on the path T' constructed in the reduction in Theorem 4 such that $R_v = T'$. The rest is exactly as above. \square

4.3 Parametrized Cases

Unlike in Section 3, parametrization by number of pre-drawn subtrees k is mostly not helpful. Every problem is already NP-complete for $k = 0$ or $k = 1$ with exception of $\text{REPEXT}(\text{PATH}, \text{ADD})$ for which Proposition 3 straightforwardly generalizes.

Similarly, the low number of components c does not make the problem any easier. We can easily modify the above reductions to show that all problems remain NP-complete even if the graph G is connected.

Concerning size of the tree, Proposition 4 straightforwardly generalizes:

Proposition 7. *Let t be the size of T' . The problems $\text{REPEXT}(\text{PATH}, \text{FIXED})$ and $\text{REPEXT}(\text{CHOR}, \text{FIXED})$ are fixed-parameter tractable with respect to t . They can be solved in time $\mathcal{O}(n + m + f'(t))$ where $f(t) = 2^{t^2}$.*

Proof. Proceed exactly as before, prune the graph and test all possible assignments. The only difference is that T has at most 2^t subtrees. \square

We note that a more precise bound for the number of subtrees could be used but we do not care for complexity that much.

5 Conclusions

In this paper, we have considered different problems concerning extending partial representations of chordal graphs and their three subclasses. One of the main goals of this paper is to stimulate future research in this area. Therefore, we conclude with two open problems.

The first problem concerns the only open case in the table in Figure 2.

Problem 1. What is complexity of $\text{REPEXT}(\text{PATH}, \text{BOTH})$?

Concerning parametrized complexity, we believe it is useful to first attack problems related to interval graphs. This allows to develop tools for more complicated chordal graphs. A generalization of Theorem 1 and Corollary 2 for INT seems to be particularly interesting. The PQ-tree approach seems to be a good starting point.

Problem 2. Does $\text{REPEXT}(\text{INT}, \text{FIXED})$ belong to XP with respect to k where k is the number of pre-drawn intervals?

References

1. Bläsius, T., Rutter, I.: Simultaneous PQ-ordering with applications to constrained embedding problems. CoRR abs/1112.0245 (2011)
2. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and planarity using pq-tree algorithms. *Journal of Computational Systems Science* 13, 335–379 (1976)
3. Corneil, D.G., Kim, H., Natarajan, S., Olariu, S., Sprague, A.P.: Simple linear time recognition of unit interval graphs. *Information Processing Letters* 55(2), 99–104 (1995)
4. Corneil, D.G., Olariu, S., Stewart, L.: The LBFS structure and recognition of interval graphs. *SIAM Journal on Discrete Mathematics* 23(4), 1905–1953 (2009)

5. Deng, X., Hell, P., Huang, J.: Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs. *SIAM J. Comput.* 25(2), 390–403 (1996)
6. Fulkerson, D.R., Gross, O.A.: Incidence matrices and interval graphs. *Pac. J. Math.* 15, 835–855 (1965)
7. Garey, M.R., Johnson, D.S.: Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing* 4(4), 397–411 (1975)
8. Gavril, F.: A recognition algorithm for the intersection of graphs of paths in trees. *Discrete Mathematics* 23, 211–227 (1978)
9. Golumbic, M.C.: *Algorithmic Graph Theory and Perfect Graphs*. North-Holland Publishing Co. (2004)
10. Jampani, K., Lubiw, A.: The simultaneous representation problem for chordal, comparability and permutation graphs. In: *Algorithms and Data Structures, Lecture Notes in Computer Science*, vol. 5664, pp. 387–398 (2009)
11. Jansen, K., Kratsch, S., Marx, D., Schlotter, I.: Bin packing with fixed number of bins revisited. In: *Algorithm Theory - SWAT 2010, Lecture Notes in Computer Science*, vol. 6139, pp. 260–272 (2010)
12. Klavík, P., Kratochvíl, J., Krawczyk, T., Walczak, B.: Extending partial representations of function graphs and permutation graphs. Accepted to *ESA 2012*, track A (2012)
13. Klavík, P., Kratochvíl, J., Otachi, Y., Ignaz, R., Saitoh, T., Saumell, M., Vyskočil, T.: Extending partial representations of proper and unit interval graphs. In preparation. (2012)
14. Klavík, P., Kratochvíl, J., Vyskočil, T.: Extending partial representations of interval graphs. In: *Theory and Applications of Models of Computation - 8th Annual Conference, TAMC 2011, Lecture Notes in Computer Science*, vol. 6648, pp. 276–285 (2011)
15. Looges, P.J., Olariu, S.: Optimal greedy algorithms for indifference graphs. *Comput. Math. Appl.* 25, 15–25 (1993)
16. Rose, D.J., Tarjan, R.E., Lueker, G.S.: Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing* 5(2), 266–283 (1976)
17. Schäffer, A.A.: A faster algorithm to recognize undirected path graphs. *Discrete Appl. Math* 43, 261–295 (1993)